Programación Python para BigData

Lección 5: Mongo BD

eip ESCUELA INTERNACIONAL POSGRADOS

Indice

Introducción

En el tema actual se ha explicado como trabajr con mongoBD mediante consola, script y aplicación.

Se pide realizar un manual explicando el uso de MongoBD. Se hará una pequeña introducción con el empleo de la aplicación y tras ello se explicara el script creado para la resolución de la aplicación.

Uso de Robo3t

Es importante recordar que los comando de consola sugeridos en el manual son comando de sistemas Linux con lo que la realización de estos pasos en sistemas Windows puede variar.

Primeros pasos

Antes de nada, será necesario arrancar el servicio de mongoDB, para ello emplearemos docker.

Si no tenemos la imagen creada , escribiremos la siguiente línea en nuestra consola:

```
sudo docker pull mongo
```

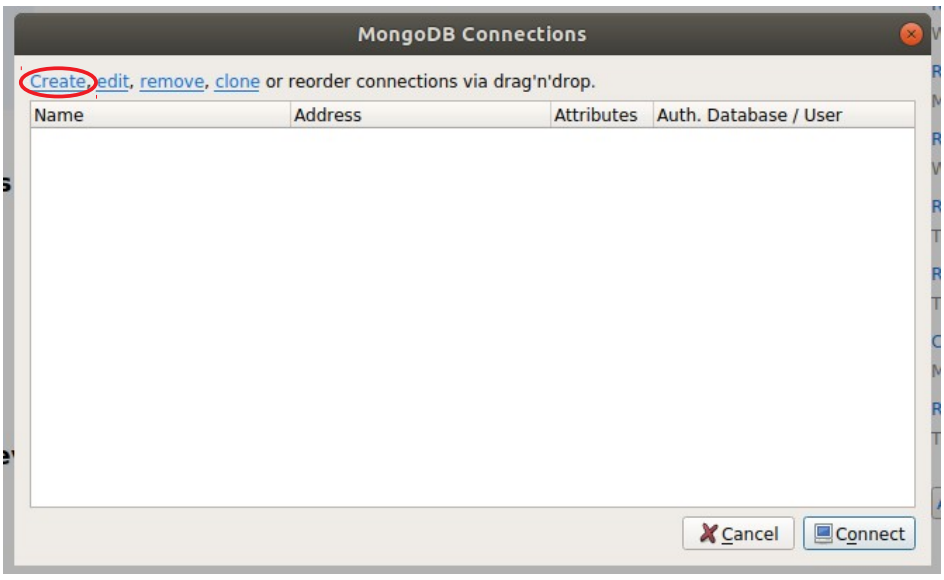
Esto creará la imagen del servicio de mongoDB en nuestro ordenador. A continuación arrancaremos el servicio con la siguiente linea:

```
sudo docker run -d --name mongoBD -p 27017:27017 mongo
```

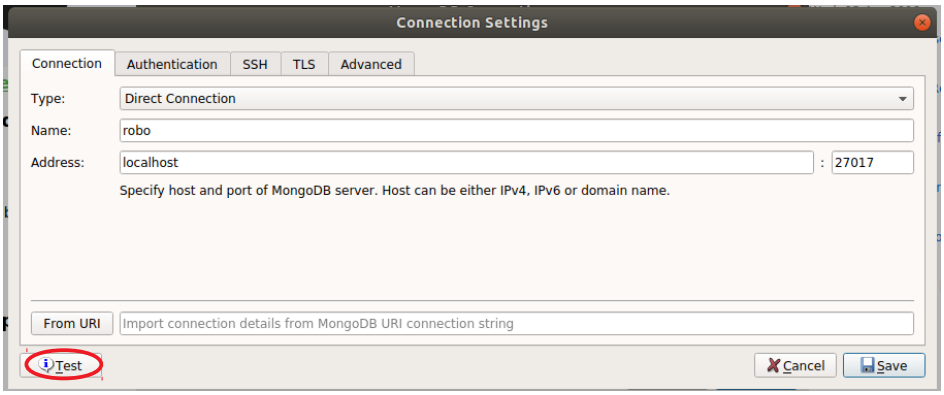
Tras arrancar el servicio, es momento de arrancar Robo3t, para ello, nos dirigiremos a la carpeta en la que tengamos los archivos de robo3t desde el terminal y lo ejecutaremos con “./robo3t”

Creando una conexión

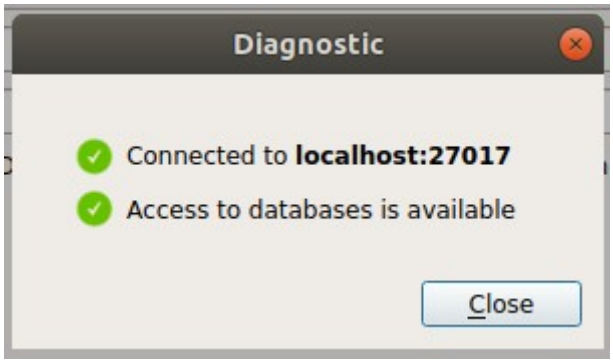
Una vez ejecutemos robo3t, aparecerá la siguiente pantalla:



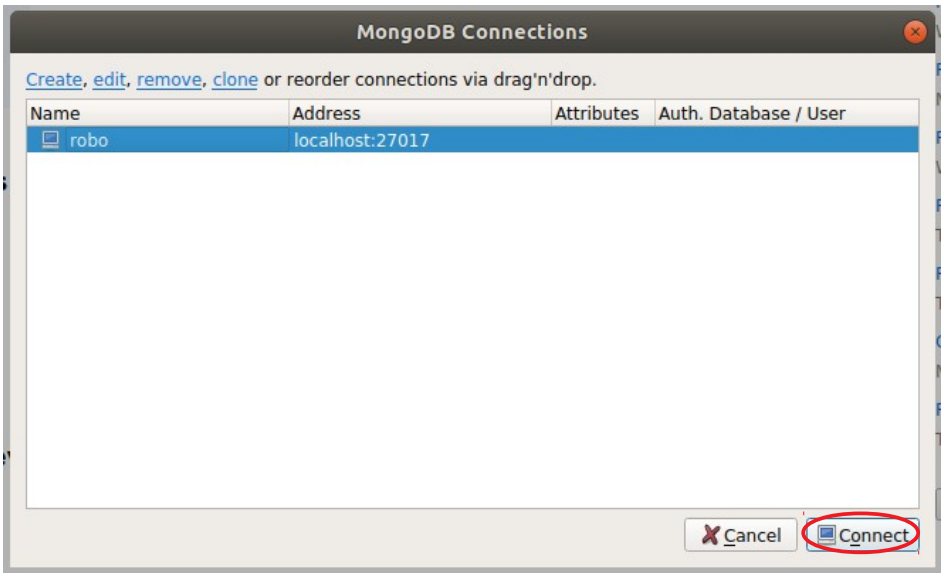
Una vez aquí, pulsaremos Create para crear una conexion y aparecerá la siguiente ventana:



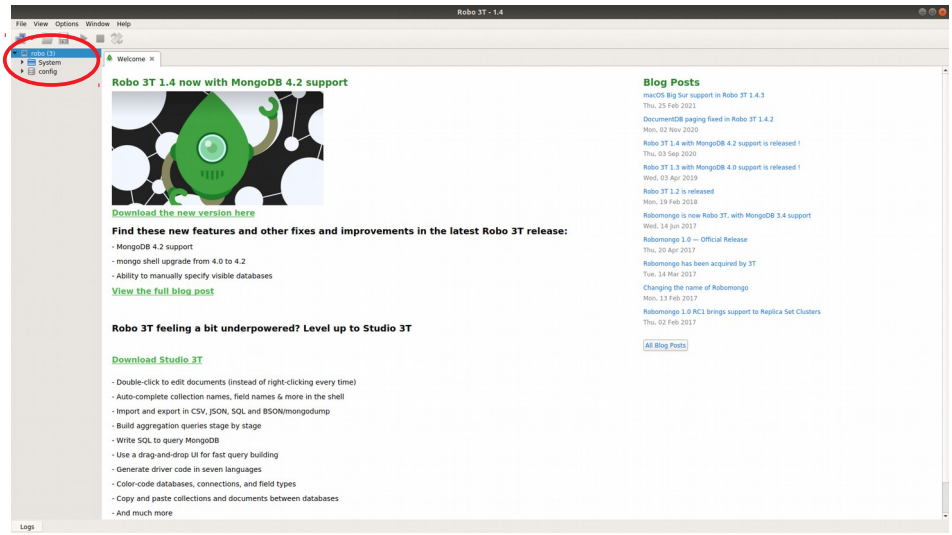
Podremos testear la conexión al servidor pulsando “Test”, si todo esta correcto aparecerá lo siguiente:



Cerraremos la ventana, le pondremos un nombre a la conexión y pulsaremos “Save”. Con esto, habremos creado nuestra base de datos, el siguiente paso será conectarnos a ella. En la siguiente ventana,



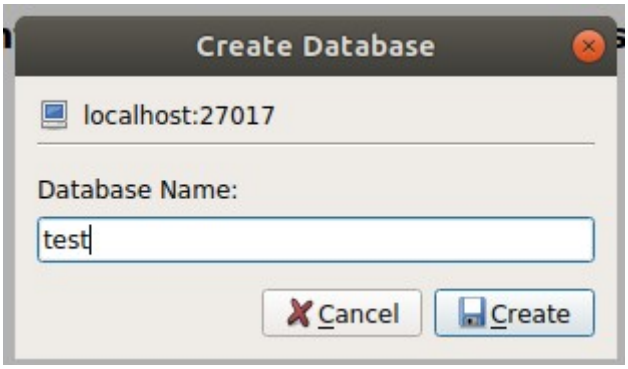
seleccionaremos nuestra base de datos y pulsaremos en connect, tras eso, nos aparecerá en el lado izquierdo de la pantalla.



A partir de aquí, prácticamente todo se hará con el ratón.

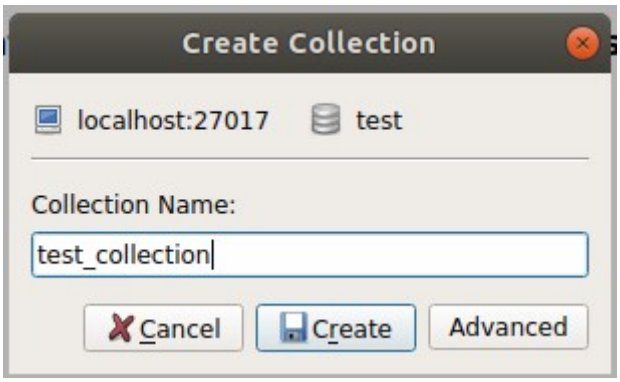
Creando la base de datos y sus datos

Para crear la base de datos haremos click con el botón derecho del ratón sobre el nombre de la conexión y seleccionaremos “Create Database”. Aparecerá la siguiente pantalla:



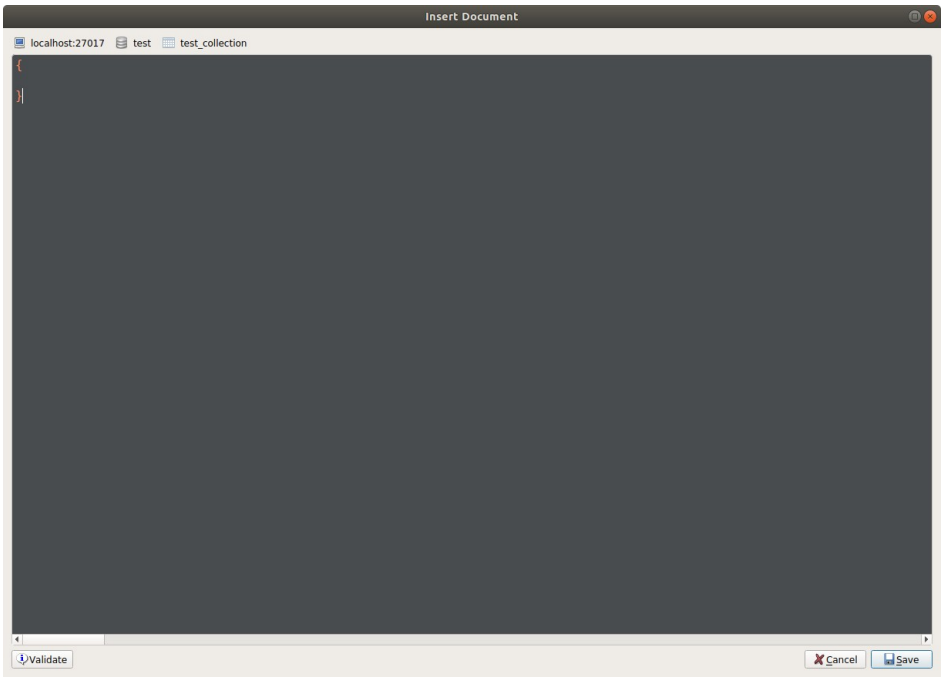
Daremos nombre a la base de datos y pincharemos en “Create”.

Una vez creada, desplegaremos la lista que se crea con la base de datos y sobre “Collections” haremos clic derecho y seleccionaremos “Create Collection...”



Le daremos un nombre y “Create”.

Para introducir datos en la colección creada, despegaremos la lista de “Collections” y sobre test_collection haremos clic derecho e “Insert Document...”



una vez aquí introduciremos los datos como si fuera un diccionario de Python, es decir:

key1: vlaue1, key2: value2, …

Daremos clic en sabe y ya habremos introducido los datos en la colección.

Para visualizarlos vasta con hacer doble clic sobre la colección, para editarlos, seleccionaremos el dato, botón derecho y “Edit Document”. Al finalizar, pincharemos en “save”.

Para editar un grupo de datos, clic derecho sobre uno de los datos, “Delete Document” y confirmaremos el mensaje, para eliminar la base de datos y la colección será lo mismo solo que seleccionaremos “Drop Collection” o “Drop Database”.

Pasemos al Script.

Uso de pymongo

Una vez mas, he optado por crear una clase y ofrecer una solución (esta vez mas) generalizada, de modo que se pueda usar con cualquier base de datos que se desee.

Estableciendo las conexiones:

```python
class Mongo:
    def __init__(self, host, port):
        # creamos la conexion con el cliente
        self.client = pym.MongoClient(host, port)
        # print(self.client)

    def conect_db(self, db):
        # establecemos la conexión a la base de datos establecida
        self.db = self.client[db]
        # print(self.db.user)

    def conect_coll(self, coll):
        # conectamos con la coleccion deseada
        self.coll = self.db[coll]
```

```python
db = Mongo('localhost', 27017)
db.conect_db('actividad')
db.conect_coll('notas')
```

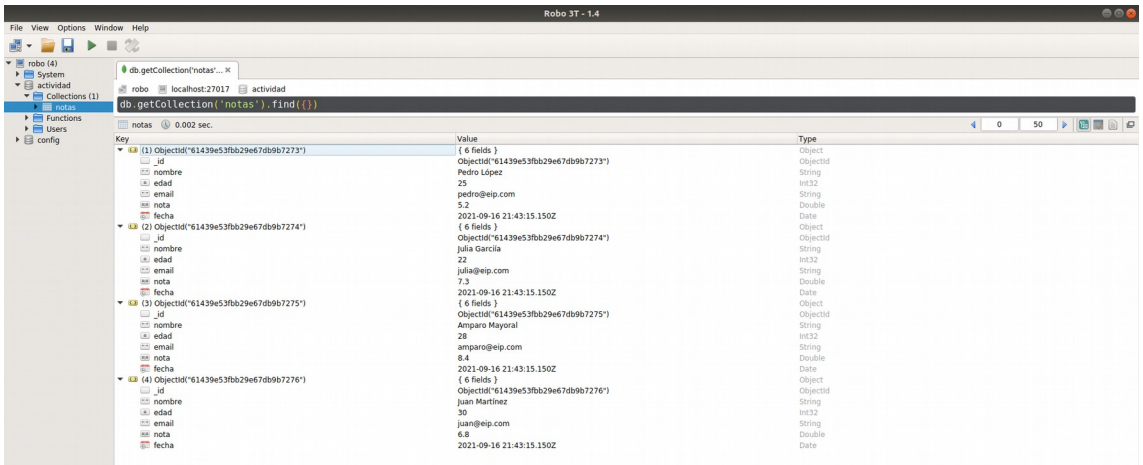
Al crear la primera instancia de la conexión tendremos que especificar el host y el puerto al que nos conectaremos, tras ello definiremos el nombre de la base de datos a crear, o a la que deseamos conectarnos y finalmente tendremos que establecer a que colección deseamos conectarnos.

Añadiendo información

```python
    def insert_info(self, info):  # info as variable & type(info) == list
        if type(info) is list:
            for elem in info:
                try:
                    self.coll.insert(elem)  # type(elem) = dict()
                except TypeError:
                    print(f'El tipo de dato de "{elem}" no es correcto.')
        else:
            print('TypeError: Introduced data must be a list.')
```

```python
info = [
    {'nombre': 'Pedro López', 'edad': 25, 'email': 'pedro@eip.com',
     'nota': 5.2, 'fecha': datetime.now()},
    {'nombre': 'Julia Garciía', 'edad': 22, 'email': 'julia@eip.com',
     'nota': 7.3, 'fecha': datetime.now()},
    {'nombre': 'Amparo Mayoral', 'edad': 28, 'email': 'amparo@eip.com',
     'nota': 8.4, 'fecha': datetime.now()},
    {'nombre': 'Juan Martínez', 'edad': 30, 'email': 'juan@eip.com',
     'nota': 6.8, 'fecha': datetime.now()}
]
db.insert_info(info)
```

Al emplear este función podrá introducirse un único dato o una colección completa de datos, pero en cualquiera de ambos casos, tendrá que tener el formato de una lista como se muestra en el ejemplo. El resultado en robo3t será el siguiente:



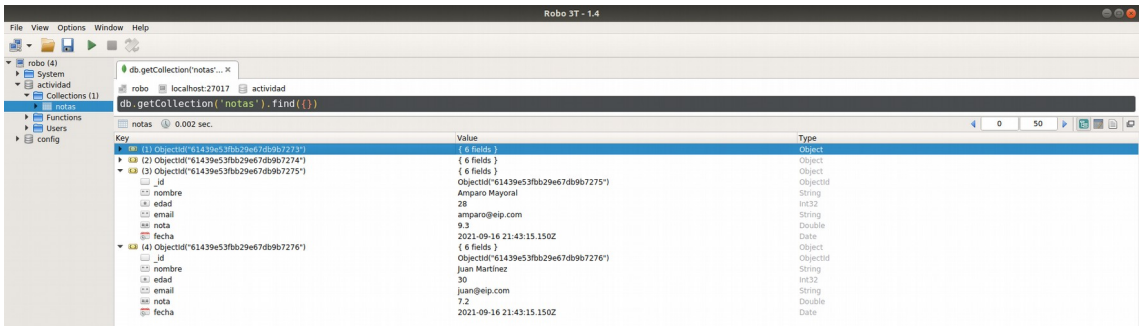
Actualizando la información existente

```
    def update_info(self, ref, info):  # info == {key: value}
        if type(ref) is dict:
            self.coll.update(ref, {'$set': info})
        # recorremos los datos para buscar la referencia
        else:
            for dato in self.coll.find({}):
                for key in list(dato.keys()):
                    if dato[key] == ref:
                        self.coll.update({key: ref}, {'$set': info})
```

```
update = [
    ({'nombre': 'Amparo Mayoral'}, {'nota': 9.3}),
    ('Juan Martínez', {'nota': 7.2})
]
for elem in update:
    db.update_info(elem[0], elem[1])
```

A la hora de actualizar la información de la colección, podremos o bien pasar el par {clave: valor} con la referencia que queremos actualizar o directamente introducir la referencia del dato que deseamos actualizar, pero siempre tendrá que ser ser una tupla con la forma (ref, {key: newValue}).

El resultado del uso de esta función es el siguiente:



Imprimiendo las colecciones

```
    def print_collection(self):
        for dato in self.coll.find({}):
            print(dato)
```

En este caso la función se encarga de dar un formato sencillo a la salida para poder leer la información de forma cómoda dando el siguiente resultado:

```
(BigData) elidas@FireBall:~/Escritorio/Master/8-Programacion_para_BigData/LEccion_5-MOngoBD$ python mongodb_OscarGutierrez.py
{'_id': ObjectId('6143b12698ad4379c143c1bc'), 'nombre': 'Pedro López', 'edad': 25, 'email': 'pedro@eip.com', 'nota': 5.2, 'fecha': datetime.datetime(2021, 9, 16, 23, 3, 34, 352000)}
{'_id': ObjectId('6143b12698ad4379c143c1bd'), 'nombre': 'Julia Garcíia', 'edad': 22, 'email': 'julia@eip.com', 'nota': 7.3, 'fecha': datetime.datetime(2021, 9, 16, 23, 3, 34, 352000)}
{'_id': ObjectId('6143b12698ad4379c143c1be'), 'nombre': 'Amparo Mayoral', 'edad': 28, 'email': 'amparo@eip.com', 'nota': 9.3, 'fecha': datetime.datetime(2021, 9, 16, 23, 3, 34, 352000)}
{'_id': ObjectId('6143b12698ad4379c143c1bf'), 'nombre': 'Juan Martínez', 'edad': 30, 'email': 'juan@eip.com', 'nota': 7.2, 'fecha': datetime.datetime(2021, 9, 16, 23, 3, 34, 352000)}
```

Cribando Información

```
    def find_group(self, ref, min, max):
        group = list()
        for dato in self.coll.find({}):
            if min <= dato[ref] and dato[ref] <= max:
                group.append(dato)
        return group
```

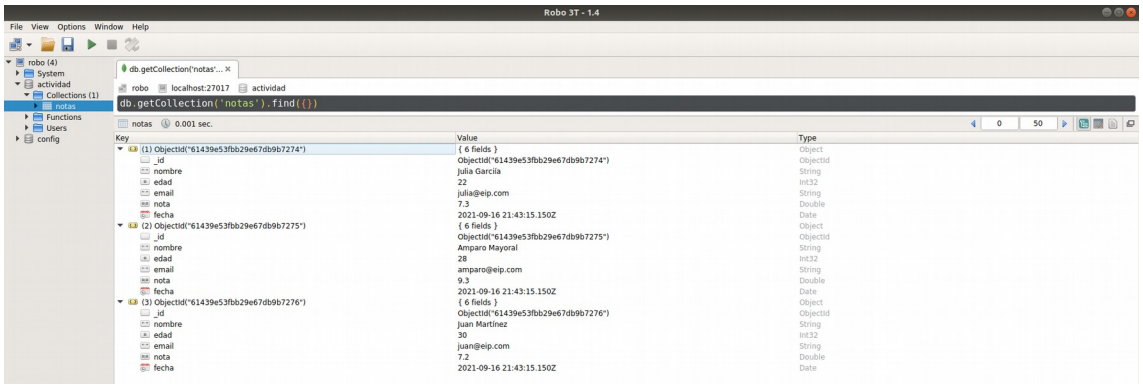
Esta funcion requiere de tres valores, la referencia que se usara para cribar los datos, el valor mínimo que podrá tener el dato y el valor máximo. Devolverá una lista con la información obtenida:

```
elidas@FireBall: ~/Escritorio/Master/8-Programacion_para_BigData/LEccion_5-MOngoBD
Archivo Editar Ver Buscar Terminal Ayuda
(BigData) elidas@FireBall:~/Escritorio/Master/8-Programacion_para_BigData/LEccion_5-MOngoBD$ python mongodb_OscarGutierrez.py
{'_id': ObjectId('61439e53fbb29e67db9b7274'), 'nombre': 'Julia Garcíia', 'edad': 22, 'email': 'julia@eip.com', 'nota': 7.3, 'fecha': datetime.datetime(2021, 9, 16, 21, 43, 15, 150000)}
{'_id': ObjectId('61439e53fbb29e67db9b7276'), 'nombre': 'Juan Martínez', 'edad': 30, 'email': 'juan@eip.com', 'nota': 7.2, 'fecha': datetime.datetime(2021, 9, 16, 21, 43, 15, 150000)}
```

Eliminando datos

```
def delete_info(self, info):
    if type(info) is dict:
        self.coll.delete_one(info)
    # recorremos los datos para buscar la referencia
    else:
        for dato in self.coll.find({}):
            for key in list(dato.keys()):
                if dato[key] == info:
                    self.coll.delete_one({key: info})
```

Al igual que para actualizar, se podrá introducir como referencia el valor del dato que deseamos eliminar o el par {key: value} dando como resultado, el que sigue:



A tener en cuenta

Tal y como está diseñado el programa, trabaja sobre la conexión mas reciente en robo3t por así decirlo por lo que habrá que tener cuidado de no estropear la conexión indebida cuando trabajemos con este script.